Connectivity Graph Reconstruction for Networking Cloud Infrastructures

Pernelle Mensah, Samuel Dubus, Wael Kanoun, Christine Morin, Guillaume Piolle, Eric Totel

Connectivity Graph Reconstruction for Networking Cloud Infrastructures

Pernelle Mensah*†‡, Samuel Dubus* and Waël Kanoun*
*Cybersecurity Paris Saclay
Nokia Bell Labs
Nozay, France
firstname.name@nokia-bell-labs.com

Christine Morin†
†Myriads
Inria
Rennes, France
christine.morin@inria.fr

Guillaume Piolle‡† and Eric Totel‡†
‡CIDRE
CentraleSupélec
Cesson-Sévigné, France
firstname.name@centralesupelec.fr

***Abstract*—Cloud providers have an incomplete view of their hosted virtual infrastructures managed by a Cloud Management System (CMS) and a Software Defined Network (SDN) controller. For various security reasons (e.g. isolation verification, modeling attack paths in the network), it is necessary to know which virtual machines can interact via network protocols. This requires building a connectivity graph between the virtual machines, that we can extract with the knowledge of the overall topology and the deployed network security policy. Existing methodologies for building such models for physical networks produce incomplete results. Moreover, they are not suitable for cloud infrastructures due to either their intrusiveness or lack of connectivity discovery. We propose a method to compute the connectivity graph, relying on information provided by both the CMS and the SDN controller. Connectivity can first be extracted from knowledge databases, then dynamically updated on the occurrence of cloud-related events. This approach shows an exact, complete and up-to-date connectivity graphs computation on a representative infrastructure, in reasonable time.**

I. Introduction

Enterprise networks complexity renders the knowledge of the connectivity an essential tool to facilitate management tasks and enhance network security. The connectivity reveals the links and dependencies between available machines. That interconnection knowledge can provide useful indications to speed up the identification of points of failure in case of network outage, and can also be used to perform proactive impact analysis of devices or links failure on the users. From a security standpoint, it represents one of the building blocks for an attack graph generation, used to perform vulnerability chains construction based on network connectivity and vulnerabilities pre-requisites [13], [17], [18]. In spite of the improvement derived from exploiting them, topology and connectivity information are rarely available as a package to the administrators. Considering the Cloud, difficulties arise given the infrastructure scale, and its dynamic nature. Even if management tools allow the online collection of useful data regarding the topology, the construction of an up-to-date connectivity remains a challenge, especially with tools presenting conflicting captures of the interconnection. In physical infrastructures, the acquisition of the topology has been vastly addressed in existing works with either passive or active discovery methods. Those methodologies present issues such as their intrusiveness, extensive probing traffic and path redundancy due to the repeated interrogation of the same interfaces across several queries. With the Cloud, in which virtualization attacks and virtual infrastructure dynamic nature are introduced, new methods need to be developed. Indeed, the dynamic and customizable nature of virtual machines deployed in the Cloud, renders difficult the installation of agents dedicated to topology retrieval. On the other hand, given the Cloud economic model, providers benefit from optimizing traffic consumption, since resources are charged according to usage, hence probing traffic should be minimal. Consequently, the intrusiveness and bandwidth consumption incurred by existing approaches in physical networks represent a hindrance in the Cloud. To the best of our knowledge, we present in this paper the first method to address the retrieval of an up-to-date topology and connectivity in cloud environments as follows:

1) We designed a module using cloud technologies to retrieve an updated topology and connectivity, while avoiding limitations of physical infrastructure methods.
2) In addition to a Cloud Management Software (CMS) and to anticipate later needs for dynamic network configuration, we consider an environment comprising an SDN (Software-Defined Networking) controller interfaced with the CMS. Required information for topology and connectivity being retrieved from these two sources, we handle occasional conflicts in network states.
3) Our approach encompasses two steps: Firstly, when plugged into a running cloud environment, the module retrieves the current topology and builds the associated connectivity. This represents the **static topology and connectivity retrieval**. Secondly the module listens to change events generated inside the infrastructure and within the SDN controller in order to update the topology and connectivity previously built during the static steps: this represents the **dynamic topology and connectivity retrieval**.
4) We performed an experimental evaluation of the presented method to determine its correctness and performance in a realistic context, considering CPU and RAM consumption, the volume of data generated, and execution time for the different portions of the algorithm involved.

Section II presents the state of the art relative to topology extraction in regular infrastructures, and Section III, the environment, its model and our solution's implementation. Section IV is dedicated to the evaluation and validation, and Section V to the conclusion.

II. State of the Art on Topology Discovery

We present the state of the art regarding topology discovery techniques, organized into passive and active methods. The latter approach comprises agent-based methods, with agents installed in each device, and monitor-based methods, less intrusive due to the use of dedicated servers. Passive and active methodologies applicability to the Cloud is analyzed to uncover limits and requirements for an efficient topology and connectivity retrieval. Focus is on topology, i.e. the architecture, since limited details, if any, are given for connectivity retrieval approaches in the studied papers, i.e the protocols and direction of communication between machines.

A. Passive Discovery Methods

Passive methods are based on a non-intrusive observation of the network traffic to detect devices and reconstruct equipment topology. Passive measurements can be carried out by the deployment of specialized hardware such as network taps [10] at strategic locations in the network and binding them to traffic analyzers, however with significant costs. Other methodologies involve port mirroring, incurring an additional workload on the switches concerned, as each packet on the monitored ports is copied and sent to a monitoring host [19]. Flow export protocols such as sFlow or NetFlow can also be leveraged to reconstruct the topology. They provide access to information pertaining to layer 2, 3 and 4 of the OSI model. Few methods in the literature rely solely on a pure passive methodology for topology reconstruction. However, Eriksson and al. [9] used passive measurements to infer structural properties in the Internet. By observing the hop-count vectors between sources and passive monitors, they are able to cluster sources sharing network paths, according to similarities discovered.

B. Active Discovery Methods

There are two kinds of active methods: agent- and monitor-based approaches.

1) Agent-based Approaches: They rely on agents deployed in each device to audit, and often use the SNMP protocol, with SNMP agents installed in routers, switches or end-hosts. Network management tools allow the automated discovery of routers, subnets and layer-3 topology. Breitbart and al. [6] propose an algorithm based on standard SNMP information to construct layer-2 topology. They rely on local address forwarding tables collected in the SNMP Management Information Base (MIB) of the equipment. Lowerkamp and al. [14] have extended this work by integrating incomplete database knowledge and non-cooperative (without SNMP) equipments such as hubs. Even when relying on standard protocols such as SNMP, difficulties arise due to vendors specificities. Indeed, implementation can be extended across platforms, and inconsistencies in table indexing schemes may occur. This leads to additional challenges when processing data originating from multiple sources. Besides, agent-based approaches lead to intrusiveness into infrastructure devices, due to the need for an agent in customers' equipment.

2) Monitor-based Approaches: Monitor-based approaches are more flexible than the previous ones: they use a dedicated set of probing hosts, responsible for performing topology acquisition by leveraging protocols and applications already available in the users' devices (i.e. ICMP, traceroute, ping). They do not require the use of customers' resources, since they are independent from their hardware. Skitter [16], tool developed by the Center for Applied Internet Data Analysis (CAIDA), and the Test Traffic Measurement (TTM) [11] from RIPE Network Coordination Center (NCC) are extensive tracing systems that have been used for Internet topology discovery at the IP level. They leverage traceroute mechanisms between 24 to 200 monitors to reconstruct the topology. Donnet and al. [8] determined that traceroute-based tools for discovery can be inefficient, as they have to deal with the redundancy induced from repetitively probing the same interfaces. Hence, to decrease probing traffic, they opted for Doubletree [8], an algorithm allowing to reduce simultaneously intra- and inter- monitor duplicated data, by starting the probing at an intermediate distance between monitor and destination, and performing backward (destination-rooted tree) and forward (monitor-rooted tree) probing schemes. Monitors share paths they already probed to their destinations, to avoid their peers to take the same ones. In [7], they introduced the use of Bloom filters to reduce communication overhead induced by this path sharing methodology and proposed to limit the number of monitors to a given destination by applying clustering.

C. Challenges Faced in the Cloud

Since monitors need to be located on the sources' path in passive discovery methods, this approach is hardly adaptable to the Cloud in which communication between VMs located on the same hypervisor occur, without reaching the physical network, leading to a knowledge gap in the topology discovery. On the other hand, active discovery methods tend to impose a heavy load on the network, incurring non-billable bandwidth consumption and producing traffic potentially flagged as malicious, which brought down to the Cloud scale is not desirable. Agent-based methods are not suitable given the need to install an agent on each device, especially when this virtual machine is not directly under the control of the cloud provider. Traditional methods can only detect whether a machine is active or not, however, in the Cloud, virtual machines can be in several states: paused, shutdown, in migration, hence impacting the topology representation. Additionally, with the multi-tenant nature of the Cloud, they are not able to attribute each machine to their owner, which is a crucial addition in a security context. These limitations from traditional methodologies can be addressed in the Cloud by leveraging a centralized store containing all the needed information for the topology

reconstruction. This use of a centralized store to obtain a knowledge base in the context of cloud infrastructures has been addressed in following related works, while not directly used toward topology and connectivity retrieval. Indeed, Madi and al. [15] focus on virtualized infrastructures and tackle the verification of compliance properties such as the co-residency, co-ownership or virtualized ports consistency. They analyze data sources coming straigth from the virtualized environment, compared with data from the Openstack platform and an SDN controller to check proper instantiation. Bleikertz and al. [5] [4] aim to validate the correctness of instances configuration from an isolation perspective in the context of the Cloud. A flow analysis tool based on the extraction of virtual systems' configuration via a number of probes, and its transformation into a graph model is introduced. Based on generic or user trust assumptions, the model is augmented with traversal rules, resulting in a representation allowing to identify unwanted information flows (isolation breach), in the infrastructure. A differential analysis is performed when a change occurs, by comparing the newly obtained model and the policy to detect potential failures. The goal of these works and ours differs. While the same tools are used in our setting (i.e. Openstack and SDN controller) and Madi and al.'s, rather than aiming to verify configuration correctness across the diverse layers of the Cloud and remaining at the level of the topology as is done in this work, we plan to retrieve both the topology and the connectivity in real time and represent them in an exploitable format. Besides, their choice of processing the data retrieved in batch mode distances us from the real time property we expect from a connectivity builder. On the other hand, Bleikertz and al. consider the information flow in the infrastructure and address the dynamic evolution of their analysis. However the probes introduced for data retrieval are hypervisor-specific, in an environment without SDN controllers.

III. Building the Connectivity in a Cloud Environment

To retrieve the infrastructure's connectivity, we define the context considered. We then introduce the resulting challenges incurred, and an environment model to help the design of the implemented algorithm.

A. Context

We consider a cloud infrastructure in which we adopt the standpoint of the cloud provider. Networking is handled by an SDN controller for dynamic network configuration. This interaction is implemented via an existing application in the SDN controller, responsible for configuring the network as defined by the CMS. As a result, the CMS network configuration view is contained in the SDN controller. While the CMS presents management interfaces to both the cloud provider and the tenants, the SDN controller is exclusively managed by the cloud provider. The administration of the virtual infrastructures is delegated to the CMS, which interacts with the SDN controller to provision the tenants' networks. Multi-tenancy allows each tenant to have its own virtual infrastructure made up of a set of VMs interconnected by virtual networks. Beyond the scalability and volatile nature of the tenants' infrastructure which are characteristics of the Cloud, and need to be considered in the solution, the combination of CMS and SDN poses an additional challenge. Indeed, SDN enables administrators to deploy applications in the SDN controller. Those applications can then, according to the programmed logic, reactively modify the flow rules on virtual switches and directly affect the topology, without providing any feedback to the CMS. It results in inconsistent topology views between the Cloud Management System and the SDN Controller. On one hand, the CMS aggregates data necessary to build the topology and connectivity, i.e. the hypervisors and their capabilities, the virtual machines' location (physical hosts), their owners as well as the networks built by the tenants and the security rules enforced. On the other hand, the SDN controller allows to determine flow rules generated by providers' applications installed on top of it, and independent from the CMS configuration. Flow rules are equivalent to routing rules authorizing or forbidding connections between virtual machines based on traffic patterns and SDN application logic. From a network connectivity standpoint, they are the concrete realization of security policies. They are ordered according to arbitrary priorities given by the developer of the applications running on the SDN controller, resulting in a hierarchical ordering of the flow rules distributed into consecutive tables installed on the virtual switches. Starting from the first table, packets are matched against flow rules in decreasing order of priority and only the flow rule corresponding to the first match is enforced. Examples of actions can be drop, forward or jump to table. In our context, one of those applications is interfaced with the CMS, and implements the necessary flow rules with an arbitrarily defined priority. A similar behavior is observed for the other applications installed on the SDN controller. Only the flow rules installed with a higher priority than the ones installed via the SDN application interfaced with the CMS are unknown to the CMS and impact the resulting connectivity. Indeed, in case of positive match with an incoming packet, the actions requested in those rules will superseed lower priorities ones (in particular, the ones from the CMS).

B. Overview of the Connectivity Extraction Process

To obtain a consistent view of the connectivity requires using data from both the CMS and the SDN controller. We rely on the CMS to obtain its vision of the topology and the connectivity, connectivity being later modified by an identification via the SDN controller of higher priority rules installed. Addressing the discrepancies between the CMS and the SDN controller requires the following steps:

1) Building the initial topology and connectivity as viewed by the CMS, using the CMS databases;

2) In the SDN controller, leveraging the provided APIs to identify the applications interfaced with the CMS and register the priorities of the flow rules they provision;

3) Via the SDN controller, listing the flow rules installed, and retaining only the flow rules with higher priorities than

the ones identified in Step 2. Let *FR* be this collection of flow rules;

4) Processing each flow rule in *FR*, and based on the combination of layer 3 and 4 protocols data, querying the connectivity graph to obtain the endpoints and links to modify.

This results in a coherent static connectivity graph, reconciling the CMS and SDN controller views. For this view to be maintained considering flow rules changes in the SDN controller, a monitoring application reacts to every rule update, addition or removal, determining whether the flow rule should belong to *FR*. When it does, Step 4 is repeated for the concerned flow rule.

C. Cloud Environment Model

From any Cloud Management System used in virtual infrastructures, the same building blocks can be extracted to set up topology: *Hypervisor*, *Virtual Machine*, *Security Rule*, *Security Group*, *Tenant*, *Virtual Port*, *Virtual Router*, *Subnet* and *Network*. Networks and subnets refer to the virtualized context, while routers interconnect VMs belonging to distinct subnets. Security groups represent a collection of security rules that are applicable to virtual machines. They contain the IP range, port range, protocol (TCP, UDP or ICMP) and direction of the traffic authorized on the virtual machines. An additional option indicating traffic allowed considering the originating security group can also be provided. Let H, VM, SR, SG, T, VP, VR, S and N be the sets representing the collection of hypervisors (physical nodes), virtual machines, security rules, security groups, tenants, virtual ports, virtual routers, subnets and networks respectively. We define predicates classified in two categories: topology-related and connectivity-related. In their expression, $h \in H$, $\{vm,x,y\} \in VM$, $t \in T$, $secr \in SR$, $secg \in SG$, $vp \in VP$, $vr \in VR$, $s \in S$, $n \in N$. The topology-related predicates are the following:

- *instantiate(h,vm)*: means that the hypervisor h is the host of the virtual machine vm,
- *own(t,X)* where $X \in VM$ *or* $X \in S$ *or* $X \in SG$: means that the tenant t is the owner of the element X,
- *isAttachedTo(vp,X)* where $X \in VR$ *or* $X \in VM$: means that the element X is attached to the virtual port vp,
- *isLinkedTo(s,n)*: means that the network n is linked to the subnet s,
- *belongTo(s,vp)*: means that the virtual port vp belongs to the subnet s.

The connectivity-related predicates are the following:

- *contains(secg,secr)*: means that the security group *secg* contains the security rule *secr*,
- *isEnforcedOn(secg, vm)*: means that the security group *secg* is enforced on the virtual machine *vm*,
- *areConnected(x,y)*: means that the communication is possible between x and y for at least one combination of protocol, addresses and ports.

areConnected is a predicate partly deduced from the others. Two virtual machines *areConnected* if they are either on the same subnet or on subnets linked by routers, and their security rules allow communication for at least one combination of addresses, ports and protocol. Subnet information is provided by *belongTo* and *isLinkedTo*, while reasoning on security rule applicability is permitted by the predicates *isEnforcedOn* and *contains*. The content of the rules themselves is then interpreted in order to derive connectivity among virtual machines. In the rest of this paper, we model the network topology as a graph, which can be directly derived by assimilating the building blocks to typed nodes and the predicates to typed relationships. Figure 1 shows the graph model obtained. In the next section, we present environment dynamics and their modifications on a topology and connectivity graph based on this graph model.

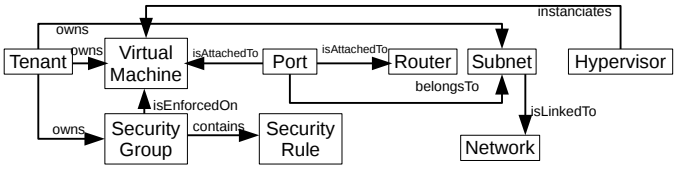

Fig. 1. Graph Model

D. Topology and connectivity graph construction

In this section, we provide more details on the topology and connectivity graph construction algorithm. Figure 2 illustrates our architecture: a cloud infrastructure administrated by a CMS and an SDN controller whose data are processed by the topology and connectivity builder. The SDN controller

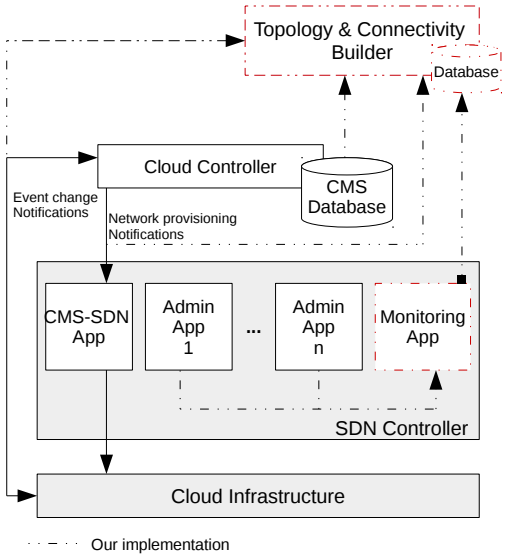

Fig. 2. Architecture

comprises an SDN monitoring application, listening to flow rule events generated by other applications. Rule characteristics are then stored in the SDN rule database. In parallel, the topology and connectivity builder runs in a dedicated server. It first creates a static topology and connectivity by processing the CMS database and stores the obtained representation into its own graph database.

Secondly, it listens to CMS-generated events to update its view. The events tracked are the creation, deletion, update and status change of virtual elements. Their impact on the topology and connectivity, considering changes occurring on virtual machines are presented in Table I. In this table, arrows represent the creation of an edge (predicate) whose type is given by the label written above. A crossed arrow represents the deletion of the predicate.

1) Static Phase: During the static phase, the aim is to establish a baseline topology and connectivity. We begin with the static topology, built with data from the Cloud Management Database. The network topology is built first, it is comprised

TABLE I
EXAMPLE OF ELEMENTARY ACTIONS PERFORMED ON VIRTUAL MACHINES IN CLOUD ENVIRONMENTS AND THEIR EFFECTS ON THE TOPOLOGY

Common actions	Effects
Create	Node creation: vm Node attribute modification: vm.state=active vm $\xleftarrow{owns}$ tenant vm $\xleftarrow{isEnforcedOn}$ secgroup vm $\xleftarrow{instantiate}$ hypervisor Generation of a port create event Node creation: port subnet $\xleftarrow{belongsTo}$ port port $\xleftarrow{isAttachedTo}$ vm vm $\xleftarrow{areConnected}$ vm_x, where vm_x is an existing machine able to communicate with vm
Delete	vm $\not\xleftarrow{owns}$ tenant vm $\not\xleftarrow{isEnforcedOn}$ secgroup vm $\not\xleftarrow{instantiates}$ hypervisor Generation of a port delete event subnet $\not\xleftarrow{belongsTo}$ port port $\not\xleftarrow{isAttachedTo}$ vm Node deletion: port Node deletion: vm
Deactivate: Pause, Shutdown, Shelve, etc.	Node attribute modification: vm.state=inactive
Activate: Unpause, Start, Unshelve, etc.	Node attribute modification: vm.state=active
Update(params)	vm.params=params

of the interactions between virtual machines, virtual ports, subnets, networks, virtual routers, security groups, security rules and tenants. In order to optimize its generation, we leverage the multi-tenant nature of the Cloud: the network topology of each tenant is independently built by concurrent threads, allowing to parallelize the task. This method allows to speed up the construction, especially when a lot of tenants are considered. Once the network topology of each tenant is obtained, the relationship is established with the cloud provider's physical infrastructure by creating an *instantiates* predicate between the tenants' virtual machines and their corresponding hypervisors. After generating the static topology (global_topology), static connectivity is obtained according to Algorithm 1. It is built by identifying groups of machines able to communicate, due to

Algorithm 1: Building static connectivity using the CMS

```
   Data: tenants_list, global_topology
   Result: CMS Connectivity in virtual infrastructure
1  foreach tenant in tenants_list do
2      routers_list = getRoutersList(tenant);
3      standaloneSubnets_list =
         getSubnetsWithNoRouter(routers_list) ;
4      foreach router in router_list do
5          virtualMachines_list =
             getVmsConnectedViaRouter(router);
6          foreach machine in virtualMachines_list do
7              securityRules_list = getSecurityRules(vm);
8              foreach sr in securityRules_list do
9                  relatedVms_list =
                     identifyRelatedVms(virtualMachines_list,
                     sr.protocol, sr.ip_dst, sr.ip_src, sr.ip_prefix,
                     sr.sec_group_id, sr.direction);
10                 createAreConnected(vm, relatedVms,
                     global_topology);
11     foreach subnet in standaloneSubnets_list do
12         virtualMachines_list = getVmsSubnet(subnet)
               /* repeat line 6 to 10                    */
```

their interconnection with routers or their belonging to a same subnet (lines 2-3). Each machine cluster is then processed to determine the effective possible communication as stated by security rules contained in the security groups they depend on (lines 8-10). This phase generates *areConnected* links as viewed by the CMS (line 10). Additionally, since flow rules provisioned by SDN applications are hierarchical, we identify the ones with a higher priority than the rules provisioned by the CMS application in the SDN controller, by querying the Flow Rules registry in the SDN controller, as shown on lines 1-4 in Algorithm 2. Indeed, these are the rules yielding discrepancies between the views from the CMS and the SDN controller. We develop a Monitoring Application installed on the SDN controller, responsible for identifying and registering these rules in a separate SDN rule database. Parameters contained in each rule allow to match the related CMS *areConnected* links and modify them according to the SDN view of the connectivity as shown on lines 6-8 in Algorithm 2.

2) Dynamic phase: During the dynamic phase, an event listener intercepts topology-related notifications generated by the Cloud Management System to store them in a queue. Events are then processed by queue consumers to update the topology and connectivity graph with the changes induced by those notifications as defined in Table I. The events are processed to modify the topology, as well as the connectivity reported in the graph. The SDN applications are also continuously monitored, in order to register any impactful modification caused by a rule creation, update or removal. The information gathered by this monitoring application allows to update an SDN rules database (see Figure 2) and modify the *areConnected* links accordingly.

Algorithm 2: Updating the static connectivity using the SDN Controller

```
Data: CMSAppsIds_list, virtualSwitches_list,
      connectivityGraphDB
Result: Merged Connectivity using CMS and SDN controller
1 CMSFlowrulePriorities_set =
   getFlowrulePrioritiesByAppsIds(CMSAppsIds_list);
2 foreach switch in virtualSwitches_list do
3     installedRules_list = getSwitchFlowrules(switch);
4     superseedingRules_list =
        getSuperseedingRules(installedRules_list,
        CMSAppsIds_list, CMSFlowrulePriorities_set);
5     foreach rule in superseedingRules_list do
6         ruleCharacteristics = getRulesCharacteristics(rule);
7         connectivityLinks =
            findMatchingPatterns(connectivityGraphDB,
            ruleCharacteristics);
8         updateConnectivityGraphDB(connectivityGraphDB,
            connectivityLinks, getRuleAction(rule));
```

E. Implementation

We describe in this section the technologies leveraged in our solution.

1) Cloud Management System: We based our solution on Openstack as our CMS, an open source software for building cloud platforms and controlling pools of compute, storage and networking resources. Information regarding the state of the platform are stored in databases by Openstack. We collect data from Keystone, Nova and Neutron databases, respectively the identity, compute and network services of the management system (Openstack). In Keystone database, we retrieve the tenants and their associated projects, a project being the canvas in which users define their virtual environments (networks, virtual machines, security groups, etc.) The information on the hypervisors and the virtual machines they host is obtained by interrogating the Nova database. The networks, subnets, routers, security groups and security rules are extracted from Neutron database.

2) SDN Controller: Given the scale of the Cloud and requirements for availability and performance, we opted for the use of ONOS (Open Network Operating System) [2], an SDN controller oriented towards service providers. It is designed to scale with the size of the network with the ability to get a cluster of controllers, hence it represents a good fit for service providers. Besides, its integration with the Openstack platform and its rich API, easing new applications development, are additional reasons motivating our choice. We leveraged its *FlowRuleService* to get information on flow rule events in virtual switches, namely updated, created or deleted flow rules. Flow rules unique identifiers allow to track their life-cycle. In flow rules, we first extract information on priority and matching patterns targeted (protocol, transport layer port, ip address, virtual switch port number, virtual switch port mac addresses, etc.). Next, we associate it to the corresponding virtual machine in Openstack. Then, we identify the treatment applied to the selected traffic (drop, allow) for that virtual machine. Both pieces of information are stored in the SDN rule database. When an event notification is made, the rule ID is used to modify, create or delete the corresponding entry.

3) Connectivity Database : Following on our graph representation of the environment, we found a suitable and flexible storage solution in graph databases. Indeed, such solutions represent a natural fit for scenarios such as network topology retrieval, as they natively store data as graphs, with edges representing relationships between typed vertices. Pattern matching and graph traversal operations allows a fast query processing. We opted for the use of Neo4j [1], an open-source graph database implemented in Java that shows interesting performance as illustrated in [12]. The use of Neo4J as database allows to build labeled vertices and edges as defined by the environment graph model (Figure 1). Furthermore, they can be augmented with additional properties. For instance, by adding a tenant id property to each concerned node, we avoid cluttering the graph with multiple ownership relationships while maintaining an efficient traversal.

IV. Evaluation

The experiments performed aim to evaluate our approach for topology and connectivity construction considering the four phases involved in the algorithm, i.e. static topology and connectivity construction, and dynamic topology and connectivity updates. Dimensions measured are the execution time, the CPU and RAM consumptions and volume of data generated by the algorithm, in order to determine the efficiency of the approach in a realistic setting. They are exclusively relative to our algorithm behavior. We identified virtual machines, routers and security rules as being the most significant parameters of the algorithm. We performed the evaluation by modifying those parameters and considering the dynamic events effect according to their types, i.e. creation or deletion. Experiments presented are implemented using the Grid'5000 testbed (https://www.grid5000.fr). 49 physical servers are used, on which Openstack Newton (CMS) is installed with 1 Controller server, 1 Network server and 45 Compute servers. A server is dedicated to ONOS, the chosen SDN controller. Another one is used for the topology and connectivity builder. Each server has 1 Intel Xeon X3440 CPU, 4 cores per CPU and 16GB of RAM.

1) Tenant's Infrastructure: To better understand the parameters' impact, we consider a cloud infrastructure containing a single tenant, with a generic 3-tier application consisting of 3 components: web, app and DB, as illustrated in Figure 3. In this tenant's architecture, the number of web servers and application servers are considered dynamic. This experiment

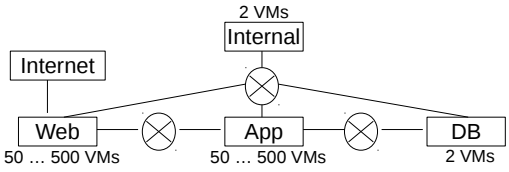

Fig. 3. Tenant's Virtual Infrastructure

is designed to assess the impact of the number of virtual machines on the dimensions introduced. We also consider the processing time to take into account the effects incurred by the deletion or creation of security rules and routers in the infrastructure. We vary the number of VMs between 100 and 1000, splitted equally among web and app servers. It accounts for a representative range, justified by the RightScale 2017 report [3] in which, a survey is made on the VM loads ran by Openstack users. As a result, the largest fraction (16%) have between 1 and 50 VMs, while only 4% have over 1000 VMs.

2) Algorithm Correctness: The verification aims to validate the correctness of the algorithm in the generation of the static connectivity, then, its ability to interpret correctly dynamic events triggered and update adequately the connectivity graph. Besides, we also verify that actions enforced by SDN flow rules are correctly translated into connectivity graph updates, hence allowing to mitigate discrepancies between CMS and SDN controller. To that end, we generate a smaller tenant infrastructure (30 VMs), on which we manually establish the connectivity. We then run the algorithm on the designed architecture and compare its output to the connectivity obtained manually. Virtual machine, router and security rule creation and deletion events are then generated. Connectivity links generated by those events via the algorithm are compared to their manually obtained counterparts. Evaluating the algorithm ability to address CMS and SDN controller is done by developing on the SDN controller an application pushing flow rules superseeding Openstack's ones and verifying the proper update of the concerned connectivity links. All these steps allow to confirm the correctness of the proposed algorithm.

3) CPU Consumption and Volume of Data Generated: Figure 4 represents the CPU consumption for the duration of experiments made to compute the connectivity, with 100 VMs and 1000 VMs. The same representation strategy is

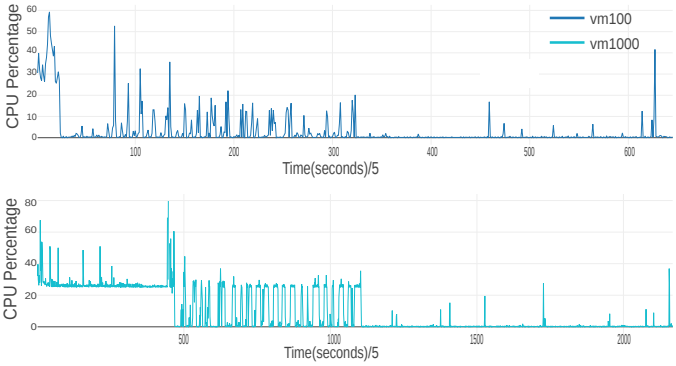

Fig. 4. Cpu Consumption for 100 and 1000 VMs infrastructures

adopted for the volume of data generated by the topology and connectivity builder in Figure 5. Given space constraints, only the downloaded data is represented, the uploaded data having a similar shape. The CPU consumption pattern is the following: first an interval, then a series of discontinuous CPU consumptions. The intervals at the beginning are an illustration of the static topology and connectivity construction of the

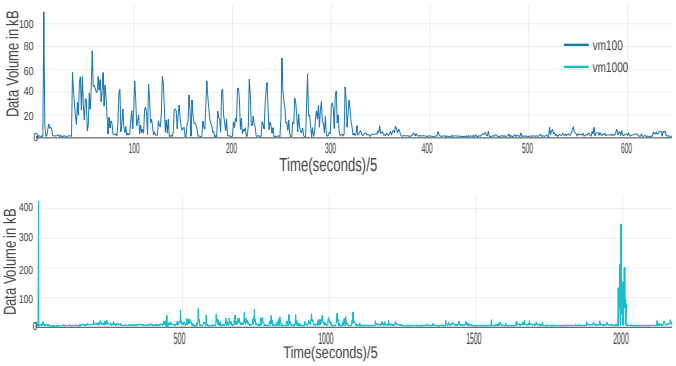

Fig. 5. Bandwidth consumption - Download link for 100 and 1000 VMs infrastructures

algorithm, while the intermittent usage illustrate the reception and treatment of an event by the algorithm, with CPU load varying according to the type of events treated. To process 1000 VMs, only 40% of CPU is requested (with some fringe values to 80%), leading us to conclude that our algorithm is efficient regarding that parameter. In Figure 5, the peak at the beginning are the requests from the builder to the CMS and SDN controller to obtain the initial data requested to build the static topology and connectivity. The smaller peaks are the queries performed on the reception of an event. Given the data volume requested (400kB max.), the impact of our approach on the CMS and the SDN controller is considered negligible.

4) Memory Consumption: For a high number of VMs, we might observe the two phases of the algorithm as illustrated in Figure 6: a peak of consumption in the beginning for the static phase, then a decrease during the dynamic phase to reach a more or less constant value. After an high memory consumption (14GB) at the beginning for the static phase, values tend to stabilize to reach 8GB for 1000 VMs. A Similar pattern is shown for 800 VMs. These values are explained by the creation and processing of a wide array of object in memory, in order to obtain the topology and connectivity of the tenant. Interestingly, for 900 VMs, the curve is similar to lower VMs amount, with a progressive RAM increase until reaching constant, and no peak at the beginning. Our server with 16 GB in total could be overloaded when considering over a thousand VMs. Hence the necessity as future work to design an algorithm that could scale out according to the load, in order to address the number of VMs.

5) Time Performance : We gain at least a factor 10 on [6] for the topology construction, with 3s against 32s. The static connectivity curve in Figure 7 shows a quadratic trend in the number of VMs, which represents an acceptable complexity for this section of our algorithm. However, the event-based processing approach we adopted frees us from successively repeating the time overhead incurred by the static phase. Indeed, the topology and connectivity is not rebuilt from scratch at each event received, but merely updated with the delta incurred. It allows to save resources but also increases execution time. Indeed, as shown in Figures 8, 9 and 10, event

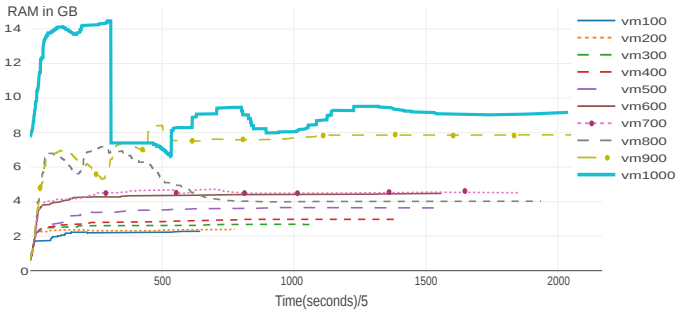

Fig. 6. Memory Consumption for different numbers of VMs (from 100 to 1000 VMs)

time processing ranges between 80ms and 12s to have an updated view of the connectivity, according to the type of event received: VM deletion is faster with roughly 100ms, while VM creation takes up to 12s for 1000 VMs. We manage to process router disconnection under 3.5s, since upon creation of the connectivity link, it is tagged with the reason for its existence, meaning router id and security rule id responsible for the connection, allowing a fast lookup of the related connected links. In the worst case, this allows to process roughly 5 events per minute for 1000 VMs.

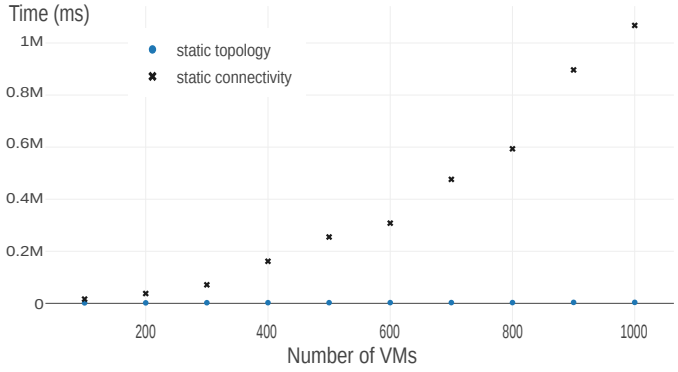

Fig. 7. Processing time for static topology and connectivity retrieval

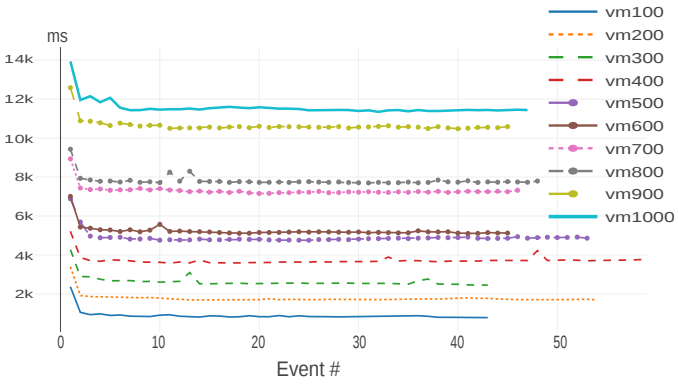

Fig. 8. Processing time of VMs creation events

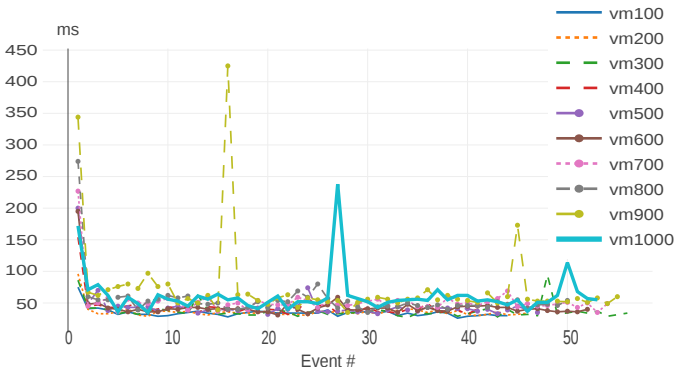

Fig. 9. Processing time of VMs deletion events

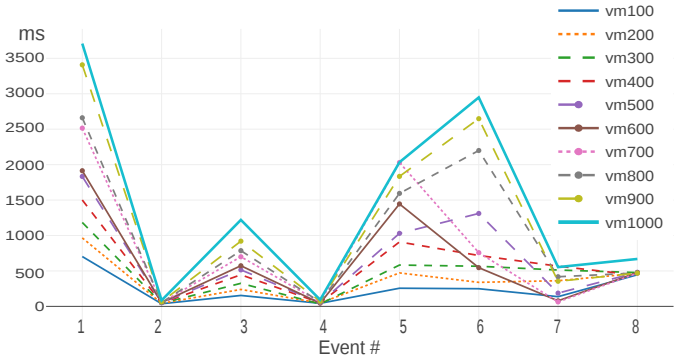

Fig. 10. Processing time of router deletion events

V. Conclusion

In this paper, we have identified the topology and connectivity extraction as a building block for risk management solutions in Cloud environments. By modeling the virtual environment and leveraging technologies available such as the CMS and the SDN controller, we designed a non-intrusive approach allowing to obtain an up-to-date view of tenants' architectures. This approach also addresses potential discrepancies between the states of the Cloud controller and the SDN controller, thus leading to an accurate representation of the connectivity. The experiments carried out illustrate the approach efficiency considering the virtual infrastructure scale. Indeed, starting from scratch, approximately 18mins is required to build the topology and connectivity for over 1000 virtual machines, without generating an excessive bandwidth for the concurrent services, or a CPU overload on the dedicated server. Modifying the graph connectivity according to change events can be done in between 100ms and 12s, according to the type of event considered, exempting us from repeating costly initializations, and allowing to process 5 modifications per minute in the worst case. On the other hand, the connectivity built using data extracted from the CMS and the SDN controller has an optimal exhaustiveness in our context. Indeed, our approach being oblivious to potential software firewalls configured by tenants in their virtual machines, it results in an over-approximation of the tenants' virtual machines connectivity. We may report configured (via CMS or SDN), but non-effective connections

between VMs due to a lack of visibility into tenants' virtual machines. However this is an acceptable approximation in a risk management context, as no potential connection link is left out of the representation. For future work, the most expensive operations lying in the establishment of the static connectivity, we aim to parallelize its construction to reduce building time, as we only did it for the static topology. Additionally, improvements could be made in the dynamic topology phase by using several consumers to process the events in the CMS queue. Due to memory consumption, a modification of the algorithm could be performed to enable scaling and load distribution to other servers. Besides, we identify interesting potential in replicating this event-based approach for maintaining attack graphs in cloud infrastructures, hence expressing the exposure needed for risk management solutions in Cloud environment.

REFERENCES

[1] Neo4j. https://www.neo4j.com/.

[2] ONOS. http://onosproject.org/.

[3] State Of The Cloud Report. Technical report, RightScale, 2017.

[4] Sören Bleikertz, Thomas Groß, Matthias Schunter, and Konrad Eriksson. Automated Information Flow Analysis of Virtualized Infrastructures. In *Proceedings of the 16th European Conference on Research in Computer Security*, ESORICS'11, pages 392–415, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Sören Bleikertz, Carsten Vogel, and Thomas Gross. Cloud Radar: Near Real-Time Detection of Security Failures in Dynamic Virtualized Infrastructures. In *Annual Computer Security Applications Conference (ACSAC)*, 2014.

[6] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz. Topology discovery in heterogeneous IP networks: the NetInventory system. *IEEE/ACM Transactions on Networking*, 12(3):401–414, June 2004.

[7] Benoit Donnet, Timur Friedman, and Mark Crovella. Improved algorithms for network topology discovery. In *International Workshop on Passive and Active Network Measurement*, pages 149–162. Springer, 2005.

[8] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 327–338. ACM, 2005.

[9] Brian Eriksson, Paul Barford, and Robert Nowak. Network Discovery from Passive Measurements. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 291–302, New York, NY, USA, 2008. ACM.

[10] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the sprint IP backbone. *IEEE Network*, 17(6):6–16, November 2003.

[11] Fotis Georgatos, Florian Gruber, Daniel Karrenberg, Mark Santcroos, Ana Susanj, Henk Uijterwaal, and René Wilhelm. Providing active measurements as a regular service for isps. In *PAM*, 2001.

[12] Salim Jouili and Valentin Vansteenberghe. An empirical comparison of graph databases. In *Social Computing (SocialCom), 2013 International Conference on*, pages 708–715. IEEE, 2013.

[13] Richard Paul Lippmann and Kyle William Ingols. An annotated review of past papers on attack graphs. Technical report, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 2005.

[14] Bruce Lowekamp, David O'Hallaron, and Thomas Gross. Topology Discovery for Large Ethernet Networks. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 237–248, New York, NY, USA, 2001. ACM.

[15] Taous Madi, Suryadipta Majumdar, Yushun Wang, Makan Pourzandi, and Lingyu Wang. Auditing security Compliance of the Virtualized Infrastructure in the Cloud: Application to OpenStack. In *6th ACM Conference on Data and Application Security and Privacy ACM CODASPY 2016*, 2016.

[16] D McRobb, K Claffy, and T Monk. Skitter: Caida's macroscopic internet topology discovery and tracking tool, 1999.

[17] S. Noel, S. Jajodia, and A. Singhal. Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1(1), 2010.

[18] X. Ou and A. Singhal. *Quantitative Security Risk Assessment of Enterprise Networks*. Springer, 2012.

[19] William Tu, Priya Thangaraj, Jui-hao Chiang, and Tzi-cker Chiueh. Automated service discovery for enterprise network management, 2009.